# Lexical Functional Grammar: Analysis and Implementation

Magdalene Grantson

*Department of Computer Science*

*Lund University, Box 118, 221 00 Lund, Sweden*

May 16, 2002

**Abstract**

Lexical Functional Grammar LFG is a theory of grammar e.i. in general a theory of syntax, morphology, and semantics. It postulates two levels of syntactic representation of a sentence, a constituent structure (c-structure) and functional structure (f-structure). LFG formalism provides a simple set of devices for describing the common properties of all human Languages and the particular properties of Individual Languages. In this paper, I will describe the basic concepts underlining Lexical Functional Grammar and will implement:

- a parser for the c-structure.
- a parser for the c-structure with functional annotations

All implementations will be done in Prolog (with the use of DCF rules and list)

## 1 Introduction

The formalism of Lexical Functional Grammar was first introduced in 1982 by Kaplan and Bresnan after many years of research in this area. It has since been applied widely to analyse linguistics phenomena. This theory of grammar assigns two levels of syntactic representation to a sentence, the constituent structure (c-structure) and functional structure (f-structure). The c-structure have the form of context-free phrase structure trees. It serves as the basis for phonological interpretation while the f-structure are sets of pairs of attributes and values. Attributes may be features, such as tense and gender, or functions, such as subject and object. The f-structure actually encodes linguistic information about the various functional reactions between parts of a sentence. Many phenomena are thought to be more naturally analysed in terms of grammatical functions as represented in the lexicon or in f-structure, rather than on the level of phrase structure. For example, the alternation in the syntactic position in which the logical object (theme argument) appears in corresponding active passive sentences has been viewed by many linguists as fundamentally syntactic in nature and treated as transformations is handled by LFG as Lexicon. Grammatical functions are not derived from phrase structure configurations, but are represented at the parallel level of functional structure. C-structure varies across languages, however f-structure representation, which contains all necessary information for semantic interpretation of an utterance, is said to be universal.

## 2 Structural Representation in LFG

There are different kinds of information dependencies constituting parts of a sentence, thus giving rise to different formal structures. LFG is basically made up of *lexical structure, functional structure and constituent structure.* There are also notational conventions of LFG that a person is likely to encounter in a grammar written in LFG. The rules of LFG contain expressions known as FUNCTIONAL SCHEMATA which are symbols that appear at the right of the $-->$ arrow. The Figure 1 below shows a format for writing rules in LFG. Examples of such a format with further explanations of the arrows will be given in the next subsections.

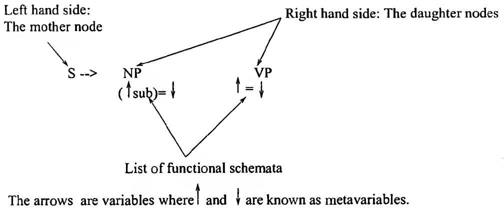
Left hand side:
The mother node
Right hand side: The daughter nodes
S --> NP VP
(↑sub)= ↓
↑ = ↓
List of functional schemata
The arrows are variables where ↑ and ↓ are known as metavariables.

Figure 1:

### 2.1 Lexical structure

Lexical structures comprise information about the meaning of the lexical item, its argument structure and grammatical functions such as subject, object, etc. associated to their appropriate argument. The verb *talk* for instance, has a predicate argument structure which is made up of an agentive argument associated with the subject and a theme argument associated with the object function.

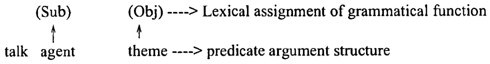
(Sub)
(Obj) ----> Lexical assignment of grammatical function
talk agent
theme ----> predicate argument structure

Figure 2:

Lexical items have functional schemata in LFG. Grammatical functions are associated with lexical items and syntactic positions by mean of annotated phrase structure rule.

They mediate between lexical and constituent structure representations. Grammatical functions are considered as an interface between lexicon and syntax. Below are examples of schematised formats of LFG LEXICAL ITEMS. The name **Magdalene**, for example, comes with the grammatical information such as gender.

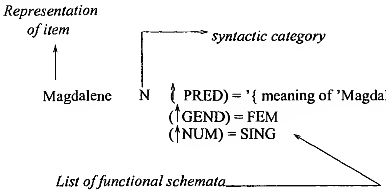

Figure 3:

A lexical rule takes a lexical item as input and returns a new lexical item. It is defines over a whole class of lexical items.

## 2.2 Constituent structure (c-structure)

The c-structure encodes hierarchical grouping and linear order. That is; it indicates the hierarchical structure of phrasal constituents in the string in a way familiar to most linguists. Functional annotations is functional schemata transfered into tree and is interpreted as deriving information about the functional structure.

In creating c-structures, all we need is context-free phrase-structure rules, phrase structure trees and the insertion of functional schemata. Language specific annotation of phrase-structure rules helps to identify the grammatical functions that occur in specific syntactic positions. Below is a set of phrase structure rules:

```
S --> NP  VP
NP --> (Determiner) N (PP)
PP --> preposition NP
VP --> V ( NP) (NP) PP
```

If we consider the sentence **Magda likes Ann**; we first of all consider the syntactic rules of this sentence, then we construct a tree with annotations prescribing the rules. Figure 4 shows a relation between rules and annotations in the tree.

The arrow $\uparrow$ Refers to the f-structure (see next subsection) of the mother node. It is instantiated *(Instantiation transforms the schemata into functional equation)* by the node immediately dominating the constituent under which the arrow is placed. The arrow $\downarrow$ refers to the f-structure of the current node. Thus from the rule S $--$ >NP VP, the equation states that NP is the subject (Sub) of S that dominates it. The $\uparrow = \downarrow$ equation

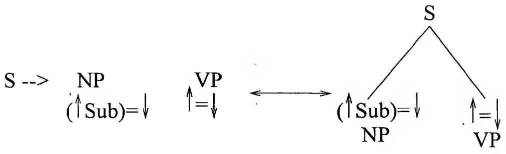

Figure 4: Relation between rules and annotations in the tree

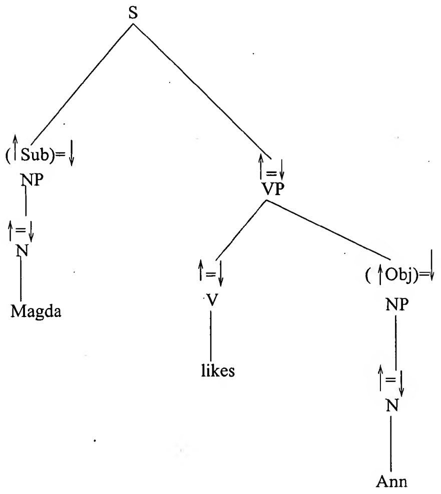

Figure 5:

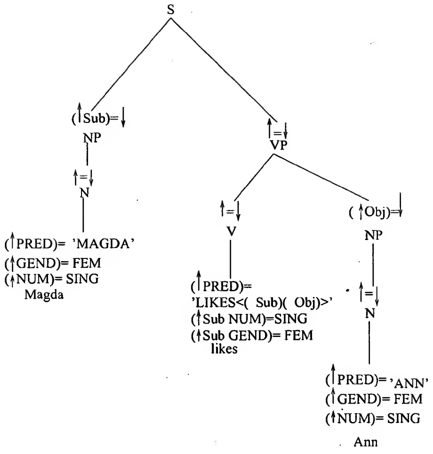

Figure 6:

beneath VP indicates that the features of that node are shared with higher nodes, making all functional information carried by this node also direct information about the mother's f-structure. ( $\uparrow$ SUB)= $\downarrow$ ) means all functional information carried by this node goes into the subject part of the mother's function. Thus from the sentence **Magda likes Ann** the next stage of the tree will be like indicated in Figure 5.

The c-structure is complete after introducing the annotations specified by the lexical entries for **Magda likes Ann**. This is achieved by consulting the lexical entry for each lexical item in the tree for their functional schemata. Figure 6 is the annotated c-structure for the sentence **Magda likes Ann**

## 2.3 Functional Structure (f-structure)

Functional structures are unification-based grammars and encode information about the various functional reactions between parts of sentences. They are themselves functions from attributes to value, where the value for an attribute can be:

- Atomic symbols eg. [NUM SING]

- Semantic form eg. [PRED 'TRUST¡( ↑ S (↑ OBJ)¿]
- f-structure, eg.

$$\left[ Sub \begin{bmatrix} PRED & 'MAGDA' \\ NUM & SING \\ GEND & FEM \end{bmatrix} \right]$$

Graphical f-structures are represented in this paper as material enclosed in large square brackets as in Section 3. There is a basic assumption in LFG, which states that there is some f-structure associated with each node in the constituent structure tree. Hence lexical information is combined with structural information available from the c-structure tree to get the f-structure.

## 3 Well-formedness conditions on F-structure

The f-structure is valid according to well-formedness condition. Well-formedness condition filter out over generation in c-structures

### 3.1 Functional Uniqueness

Functional uniqueness is also refered to as consistency. It ensures that each f-structure is a function whereby each attribute has a unique value.

*Example of consistent structure in the f-structure*

$$\begin{bmatrix} NUM & SING \\ NUM & PL \end{bmatrix} \begin{bmatrix} NUM \end{bmatrix}$$

*Example of an inconsistent structure in the f-structure*

$$\begin{bmatrix} GEND & FEM \\ GEND & MAL \end{bmatrix}$$

### 3.2 Completeness

Completeness ensures that sub-categorization requirements are met. An f-structure is not well formed if it lacks values for grammatical functions that are sub-categorized by the predicate. The example below lacks a value for the subject (Sub) and is therefore termed as incomplete.

*eats*

### 3.3 Coherence

Coherence ensures that every argument is the argument of some predicate.

## 4 Implementation

The implementation is done in Prolog, with the use of DCG rules and Prolog terms. It is based on ideas from the lecture notes from Pierre Nugues' Linguistics course and [5]. The implementation of a parser for c-structure is given at Appendix A. A list of sentences provided in Section 5 are used so as to produce an appropriate c-structure. A sentence like **I have a bag** can be queried as :

```
 ?-parser(Cstructure,[i,have,a,bag])
```

to get the appropriate c-structure:

```
L = s(np([pron(i)]), vp([v(have), np([det(a), n(bag)])])) .
```

In Appendix B, an implemented demonstration of an LFG grammar (c-structure with functional annotation) is given. The check for completeness and coherence in the f-structure is not considered in the implementation. A sentence like **Magdalene washed the orange** can be queried as :

```
?- parser(Cstucture,Fstructure,[magdalene,washed,the,orange]).
```

```
to get the c-structure and f-structure as:

Cstucture = s(np(pn(magdalene)), vp(v(washed), np(det(the), n(orange))))
Fstructure = fs(pred:wash, spec:_G600, person:third, numb:sg, tense:past,
subj:fs(pred:magdalene, spec:_G645, person:third, numb:sg, tense:_G654,
subj:_G657, obj:_G660, obj2:_G663, ajunct:_G666, pcase:_G669),
obj:fs(pred:orange, spec:the, person:third, numb:sg, tense:_G787, subj:_G790,
obj:_G793, obj2:_G796, ajunct:_G799, pcase:_G802), obj2:_G618, ajunct:_G621,
pcase:_G624)

Yes
```

## 5 Sentences

### 5.1 Sample sentences for program in appendix A

```
I have a bag.
I give the bag to Peter.
Peter sells George the bag.
I walk into an Irish pub.
```

```
I order a drink.
I like soft drinks.
I dance everyday.
I had a dance yesterday.
Jane like dates.
Jane dated George.
```

### 5.2 Sample sentences for program in appendix B

```
Magdalene washed a dress.
James washed an orange.
Magdalene washed the plates.
```

## 6 Conclusion

Lexical Functional Grammar (LFG) was first documented in 1982 by Joan Bresnan.[1] It has three levels of representation,each having different formal characterization. These are Lexical structure, Constituent structure and Functional structure. The single level of syntactic representation c-structure, exits simultaneously with an f-structure representation that integrates the information from c-structure and from lexicon.

## References

[1] Bresnan J (ed) 1982. The mental representation of grammatical relations.. MIT Pree,Cambridge, Massachusetts.

[2] Kaplan, Ronald M and Bresnan J. 1982. Lexical-Functional grammar: a formal system for grammatical representation.emph Bresnan 1982.

[3] Hopcroft, John E. J.D. Ullman. 1979. Introduction to automata theory, languages and computation. Reading mass. Addison Wesley

[4] Kaplan R. , Ronald M, Maxwell J.T. 1993. LFG Grammar Writer's Workbench. Xeror Palo Alto Research Center.

[5] Clocksin W, Mellish C., Programming in Prolog.1994, Springer

[6] Pierre Nugues, 2001 lecture notes, An outline of theories, Implementations, and applications with special consideration of English, French, and German.

Appendix A

```
/*--------------------------------------------------------------
                           PARSER

This is a predicate that passes a list of words to produces an
appropriate c-structure

--------------------------------------------------------------*/
parser(L1,L):-
        s(L1,L,[]).

/*--------------------------------------------------------------

                           GRAMMAR
--------------------------------------------------------------*/

% Sentence
s(s(NP,VP)) --> np(NP), vp(VP).

% Prepositional Phrase
pp(pp(P,NP)) --> p(P), np(NP).




% noun phrase

np(np(NP)) --> np0(NP).

np0([Pron|NP5]) --> pronoun(Pron), np5(NP5).
np0([PropN|NP5]) --> proper_noun(PropN), np5(NP5).
np0([Det|NP1]) --> det(Det), np1(NP1).
np0([Adj|NP3]) --> adj(Adj), np3(NP3).
np0([PossP|NP1]) --> possessive_pronoun(PossP), np1(NP1).

np1([N|NP5]) --> noun(N), np5(NP5).
np1([Adj|NP2]) --> adj(Adj), np2(NP2).

np2([N|NP5]) --> noun(N), np5(NP5).

np3([PN|NP5]) --> proper_noun(PN), np5(NP5).

np5([],L,L).
```

```
%verb phrase


vp(vp(VP)) --> vp0(VP).

vp0([V|VP1]) --> verb(V), vp1(VP1).

vp1([NP|VP3]) --> np(NP), vp3(VP3).
vp1([PP|VP4]) --> pp(PP), vp4(VP4).
vp1([NP|VP2]) --> np(NP), vp2(VP2).

vp2([NP|VP3]) --> np(NP), vp3(VP3).

vp3([],L,L).
vp3([Adv|VP5]) --> adv(Adv), vp5(VP5).
vp3([PP|VP4]) --> pp(PP), vp4(VP4).

vp4([],L,L).
vp4([Adv|VP5]) --> adv(Adv), vp5(VP5).

vp5([],L,L).



adj(adj(Adj)) -->
        [Adj],
        {adj(Adj)}.

adv(adv(Adv)) -->
        [Adv],
        {adv(Adv)}.

det(det(Det)) -->
        [Det],
        {det(Det)}.

noun(n(N)) -->
        [N],
        {n(N)}.

p(prep(Prep)) -->
        [Prep],
        {prep(Prep)}.

possessive_pronoun(poss_pron(her)) --> [her].
```

```
pronoun(pron(Pron)) -->
        [Pron],
        {pron(Pron)}.

proper_noun(propn(PropN)) -->
        [PropN],
        {prop_n(PropN)}.

verb(v(V)) -->
        [V],
        {v(V)}.



%  Lexicon

% Adjectives
adj(irish).
adj(soft).

% Adverbs
adv(everyday).
adv(yesterday).

% Determiners
det(a).
det(the).
det(an).

% Nouns
n(bag).
n(pub).
n(dance).
n(drink).
n(walk).
n(car).
n(competition).

% Prepositions
prep(for).
prep(in).
prep(to).
prep(into).

% Pronouns
pron(she).
pron(i).
```

```
prop_n(dates).
prop_n(drinks).

prop_n(peter).
prop_n(george).
prop_n(jane).

% Verbs
v(order).
v(dated).
v(have).
v(had).
v(give).
v(sells).
v(like).
v(likes).
v(dance).
v(walk).
```

Appendix B

```
/*-----------------------------------------------------------
                         PARSER

This is a predicate that passes a list of words to  produces an
appropriate c-structure and f-structure

-----------------------------------------------------------*/


parser(Cstruc,Fstruc,Ins):-
        s(Cstruc,Fstruc,Ins,[]).

% S --> NP, VP.
%
s(s(NP,VP),fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) -->
        np(NP,Subj),
        vp(VP,fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)).

% Noun Phrase

% NP --> Determiner, N.

np(np(Det,N),FS) -->
        det(Det,FS),
        n(N,FS).

% NP --> Proper noun

np(np(PN),FS) -->
        pn(PN,FS).

% Verb Phrase

% VP --> V.

vp(vp(V),fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) -->
        v(V,fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)).

% VP --> V, NP.

vp(vp(V,NP),fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) -->
        v(V,fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)),
        np(NP,Obj).

% V --> V, NP, NP.

vp(vp(V,NP1,NP2),fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) -->
        v(V,fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)),
        np(NP1,Obj),
        np(NP2,Obj2).


% Lexicons

det(det(the), fs(pred : Pred,
                spec: the, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [the].

det(det(a), fs(pred : Pred,
                spec: a, person : Person, numb : sg, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [a].

det(det(an), fs(pred : Pred,
                spec: a, person : Person, numb : sg, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [an].

pn(pn(magdalene), fs(pred : magdalene,
                spec: Spec, person : third, numb : sg, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [magdalene].

pn(pn(james), fs(pred : james,
                spec: Spec, person : third, numb : sg, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [james].

n(n(dress), fs(pred : dress,
                spec: Spec, person : third, numb : sg, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [dress].

n(n(orange), fs(pred : orange,
                spec: Spec, person : third, numb : sg, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [orange].
```

```
n(n(plates), fs(pred : plate,
                spec: Spec, person : third, numb : pl, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [plates].

v(v(washed), fs(pred : wash,
                spec : Spec, person : Person, numb : Numb, tense : past,
                subj: fs(pred : SPred,
                        spec : SSpec, person : Person, numb : Numb,
                        tense : STense,
                        subj : SSubj,
                        obj : SOBJ, obj2 : SObj2,
                        ajunct : SAjunct, pcase : SPCase),
                obj : Obj,
                obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [washed].
```